Performance Analysis of Memory Code

Session 409

These are confidential sessions—please refrain from filming or taking pictures.

Apple Applications

Performance Analysis of Memory Code

Dave Payne
Developer Tools Group

http://developer.apple.com/wwdc2005/

Introduction

- Shark is great for event-based performance analysis
- Other tools provide additional insight into memory issues
- Use them all to make your app shine!

What You'll Learn

- Performance analysis process
- Analyzing application memory use
- Finding memory leaks
- Debugging memory usage problems

Performance Optimization

A systematic approach

1. Identify common use cases
2. Define performance goals
   - Responsiveness, throughput, scalability
3. Establish benchmarks
4. Add measurement code
5. Track results throughout development
   - Measure several runs each time, averaging the results
   - Don’t allow performance regressions
6. Focus tuning efforts on hot spots

Example: Spotlight Importers

- All importers for a given user ID run in a single `mdimport` process
- Importers should be
  - Robust
  - Fast
  - Use minimal resources
- Test performance by importing a directory with many files
  - `mdimport -p <directory>`

Areas for Benchmarking

1. Execution time
2. Memory size
3. Disk and network activity
4. Graphics

1. Execution-Time Benchmarks

- Launch time
- Time to open documents
- Time for major operations
- CPU use when idle—should be 0.0%
- Responsiveness
  - Smooth resizing and scrolling
  - Any UI "spins" during normal use?

2. Memory-Size Benchmarks

- Static memory use
  - Private memory
  - Total virtual memory
- Dynamic memory use during operations
- Leaks

3. Disk and Network I/O Benchmarks

- Files may be local or on the network
- Watch I/O activity
  - File opens/closes
  - stat() and getattrlist() calls
  - Reads/writes
  - Paging
- Number of open files

4. Graphics

- Drawing too much?
- Live resize responsiveness
- Frame rates

You Have Tools, Use Them!

- Included with Xcode tools
  - /Developer/applications/performance tools
- Full Mac OS X support
  - Universal binaries
  - Cocoa, Carbon, BSD
  - Objective-C, C++, C
- No need to recompile for profiling
- GUI tools integrated with Xcode
  - Launch tool from Xcode
  - Double-click symbol names to view source in Xcode

Mac OS X Performance Tools

	Memory Use	Execution Behavior	Resource Use
Monitoring	top BigTop Activity Monitor	top BigTop Activity Monitor	fs_usage sc_usage Quartz Debug
Analyzing	MallocDebug ObjectAlloc Shark vmmap, heap, leaks	Shark Spin Control Thread Viewer sample	Shark OpenGL Profiler

Comparison of Memory Analysis Apps

	ObjectAlloc	MallocDebug	Shark
Call Tree of All Allocations	–	✓	✓
Dynamic Memory Use	✓	–	✓
Chart of Calling Patterns	–	–	✓
Memory Leaks	–	✓	–
Allocations by Type	✓	–	–
Object Retains/Releases	✓	–	–
View Object Data	✓	–	–
View Memory Contents	✓	✓	–

Analyzing Application Memory Use

http://developer.apple.com/wwdc2005/

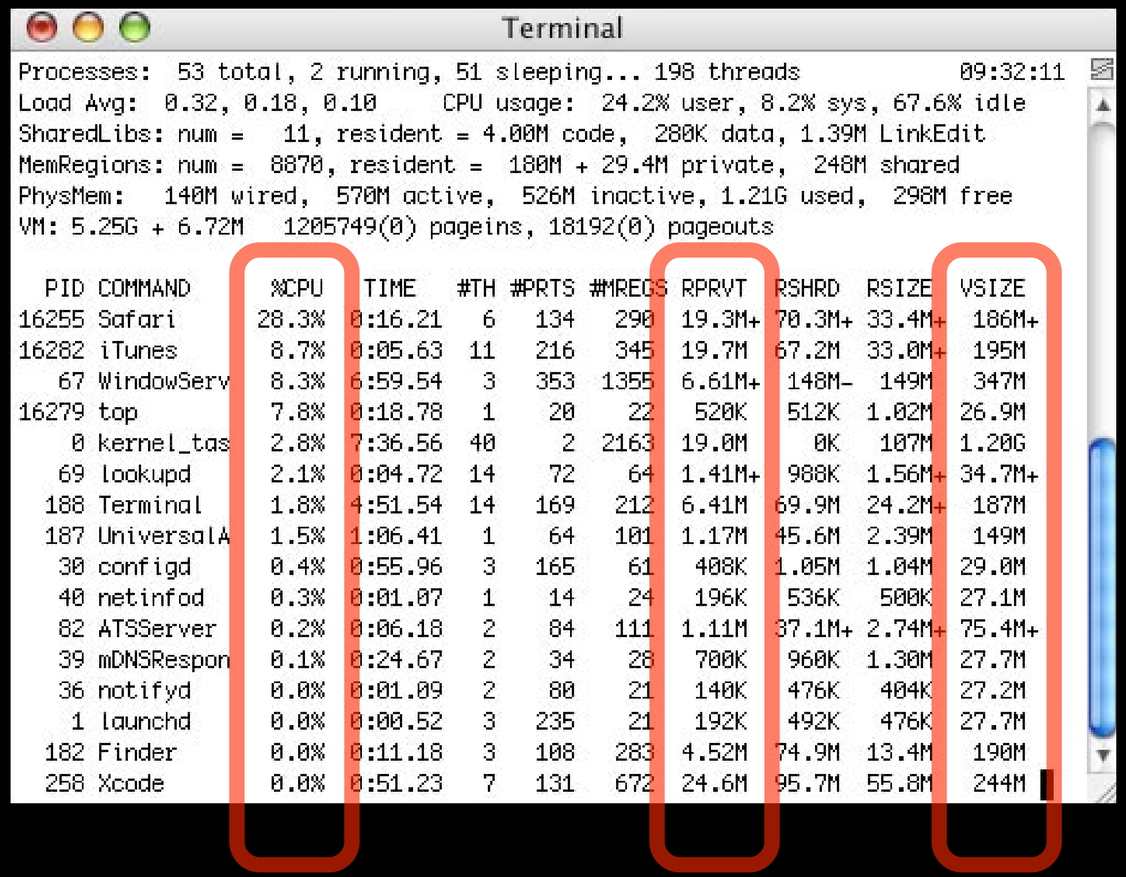
General Approach

- Define target user
  - Typical hardware configuration
  - System usage pattern
- Observe resident and virtual memory use
  - Large-memory configuration—peak memory use?
  - Small-memory configuration—any paging?
- Goals
  - Prevent paging
  - Allocate lazily
  - Eliminate repetitive allocations

Techniques

- Monitor with top, BigTop, or Activity Monitor
- Analyze memory regions with vmmap
- Trace with Shark, MallocDebug, or ObjectAlloc
- Step through code in debugger, watch resource use change

/usr/bin/top

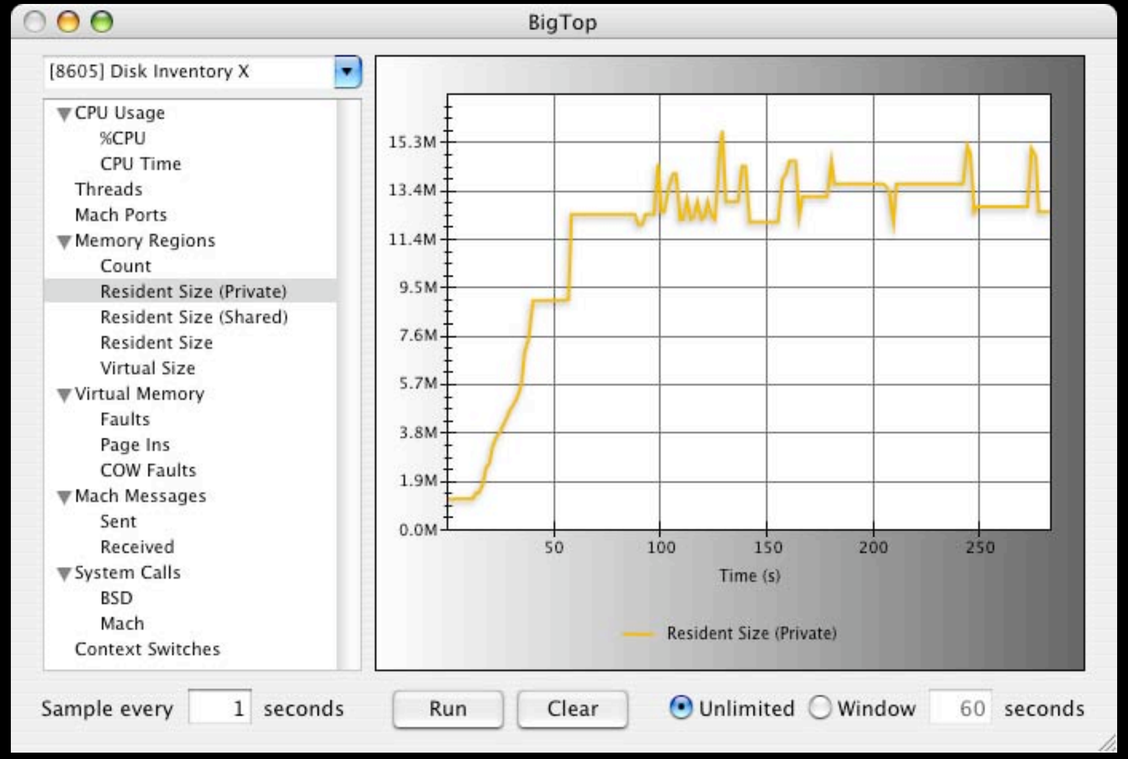


- Standard UNIX shell tool
- Ships on User CD
- Primary metrics to watch
  - % CPU—sort using `top -u`
  - Resident private size (RPRVT)
  - Virtual size (VSIZE)
- Others
  - Paging
  - Number of threads
  - Number of Mach ports

BigTop

How is resource use changing?

- Graphs performance trends over time
- Entire system or specific process
- Watch memory resident size (private) and virtual size



/usr/bin/vmmap—Memory Region Analysis

- Binary image sections
- Mapped files—now shows file names
- malloc heap
- vm_allocated regions
- Stacks

vmmap Command Line Arguments

- Basic use
  - `vmmap <app-name>`
- `-resident` shows virtual and resident sizes
- `-pages` shows sizes in pages rather than kilobytes
- `-interleaved` intersperses non-writable and writable regions

Observing Changes in VM Region Usage

- `vmmap -resident <app-name> > before.txt`
- Run operation in target application
- `vmmap -resident <app-name> > after.txt`
- `opendiff before.txt after.txt`

Demo

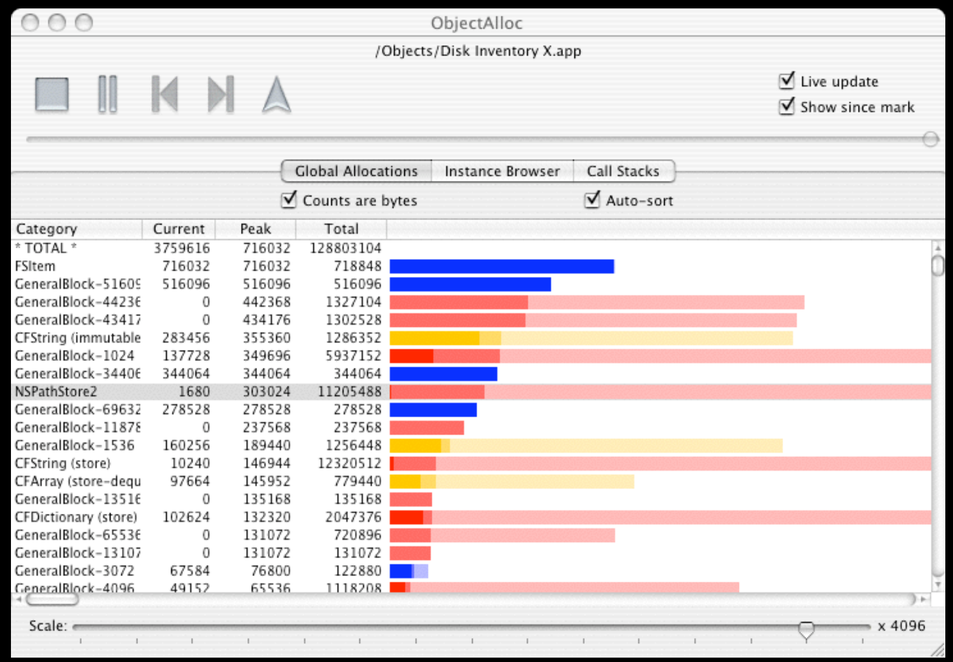
Watching VM changes with BigTop and vmmap

http://developer.apple.com/wwdc2005/

Tracing Memory Allocations with ObjectAlloc

How many objects of what types do I have?

- Cocoa, CoreFoundation objects, and malloc blocks
- Great for examining dynamic memory use
- View specific instances—size, contents, backtrace
- Can record all CF and ObjC retain and release calls



Demo

Analyzing excess object creation with ObjectAlloc

http://developer.apple.com/wwdc2005/

Finding Memory Leaks

http://developer.apple.com/wwdc2005/

Three Techniques for Finding Leaks

1. MallocDebug
2. `leaks` command with MallocStackLogging
3. `leaks` command with ObjectAlloc's reference counting

- Careful! Fixing leak on one code path can cause crashes on other paths

1. MallocDebug

- Shows call tree of which functions allocated memory
  - All allocations (existing objects only)
  - Leaks
- Now works with more apps—no longer requires flat namespace!
- Now works with fork()/exec()
  - Attach to child processes

2. `leaks` with `MallocStackLogging`

- To get stack backtraces where leaked blocks were allocated, launch app with malloc() logging enabled
  - `setenv MallocStackLogging 1`
- For more consistent results
  - Stress-test your code for long periods of time
  - Write over memory at allocation and free to eliminate “false” pointers
    - `setenv MallocPreScribble 1`
    - `setenv MallocScribble 1`

Setting UNIX Environment Variables

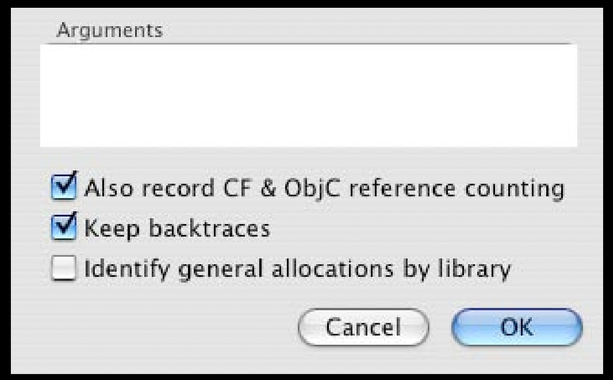
- Environment variable settings can modify behavior of performance tools
- Example: running Safari.app with MallocStackLogging set
- From `bash` session in terminal
  - `export MallocStackLogging=1`
  - `cd /Applications`
  - `./Safari.app/Contents/MacOS/Safari`
- From `tcsh` session in terminal
  - `setenv MallocStackLogging 1`
  - `cd /Applications`
  - `./Safari.app/Contents/MacOS/Safari`

Example of `leaks` Output

```
% leaks MLTEShowcase

Process 825: 6689 nodes malloced for 1567 KB

Process 825: 15 leaks for 480 total leaked bytes

Leak: 0x00669fd0  size=32       instance of 'NSURL'

        0x006022c0 0x00011380 0x00000000 0x0062d450     .`".........b.P

        0x00000000 0x00000000 0x08000100 0x00000000     ................

    Call stack: [thread 1b23]: | 0xbffff3d4 | start | start | main | NewWindow() | SetUpTheTextView
(OpaqueWindowPtr*) | TextViewDefaultSettings(OpaqueControlRef*) | AddTextToTheTextView
(OpaqueControlRef*) | CFBundleCopyResourceURL | _CFFindBundleResources |
_CFFindBundleResourcesInResourcesDir | _CFFindBundleResourcesInRawDir |
_CFSearchBundleDirectory | CFURLCreateWithFileSystemPath |
CFURLCreateWithFileSystemPathRelativeToBase | _CFURLAlloc | _CFRuntimeCreateInstance |
CFAllocatorAllocate

....
```

3. `leaks` with ObjectAlloc's Reference Counting

- Many leaks are due to retain/release mismatches
- Launch target application from ObjectAlloc
- Enable "also record CF and ObjC reference counting"
  - Tracks allocation, retain, release, and free calls for all allocations
- Exercise the target application
- In terminal, run `leaks <target-app>`
- For leaked block, find its address in ObjectAlloc's instance list
- Examine allocation, retain, and release events



Demo

Finding memory leaks

http://developer.apple.com/wwdc2005/

Debugging Memory Usage Problems

http://developer.apple.com/wwdc2005/

Types of Problems

- Buffer overrun or underrun
- Using uninitialized memory
- Accessing freed memory
- Double frees
- Memory stomping

Primary Solutions

- GuardMalloc
- malloc() environment variables
- Watchpoints in gdb

GuardMalloc Catches Memory Access Problems

- Each allocation on a separate VM page
- Unallocated guard pages before and after
- VM deallocates page when freeing
- Slow due to lots of paging
- But, forces immediate crash on bad access

Types of Problems GuardMalloc Catches

- Buffer overruns
  - Aligns allocation at end of VM page by default
- Buffer underruns
  - `setenv MALLOC_PROTECT_BEFORE 1`
- Accessing uninitialized memory
  - `setenv MALLOC_FILL_SPACE 1`
- Freeing unallocated (or previously freed) memory traps into debugger

Using GuardMalloc

- Always run in a debugger
  - From Xcode
    - Debug --> Enable GuardMalloc menu item
  - In gdb in terminal
    - `set env DYLD_INSERT_LIBRARIES=/usr/lib/libgmalloc.dylib`
- Now works with more apps—no longer requires flat namespace!
  - Don’t set `DYLD_FORCE_FLAT_NAMESPACE`
- `man libgmalloc` for full information

malloc() Environment Variables

- `man malloc` for full information
- Catching use of uninitialized memory
  - `setenv MallocPreScribble 1`
- Catching use of freed memory
  - `setenv MallocScribble 1`
- Checking heap metadata consistency
  - `setenv MallocCheckHeapStart 1` and friends
- Guard pages for large blocks
  - `setenv MallocGuardEdges 1`

Summary

- Performance analysis requires a disciplined process
- Mac OS X includes powerful tools to help you
  - Monitor and analyze memory use
  - Find memory leaks
  - Debug memory usage problems
- Use those tools to polish your apps!

More Information

http://developer.apple.com/wwdc2005/

Documentation, Sample Code, and Other Resources

http://developer.apple.com/wwdc2005/

Related Sessions

Development Tools Feedback Forum	Session 410 Friday 5:00PM

Who to Contact

Xavier Legros
Mac OS X Technology Evangelist
xavier@apple.com

Performance Tools Feedback List
perftools-feedback@group.apple.com

Xcode Lists
xcode-feedback@group.apple.com
xcode-users@lists.apple.com

Q&A

Xavier Legros
Mac OS X Technology Evangelist
xavier@apple.com

http://developer.apple.com/wwdc2005/

WWDC2005

Worldwide Developers Conference